

图文 102 BrokerContorller在启动的时候，都干了哪些事儿？

212 人次阅读 2020-02-21 07:00:00

详情 评论

BrokerContorller在启动的时候，都干了哪些事儿？



继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

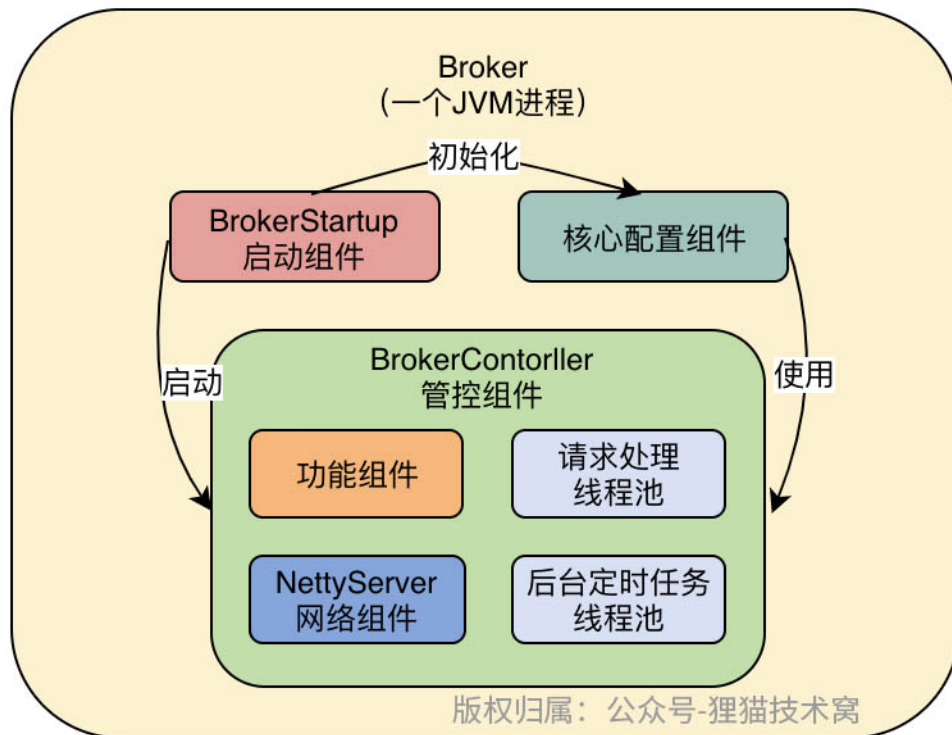
(点击下方蓝字试听)

[《从零开始带你成为MySQL实战优化高手》](#)

今天我们来给大家继续讲BrokerController的启动这块的源码。

现在BrokerController已经完成了初始化，他的用于实现各种功能的核心组件都已经初始化完毕了，然后负责接收请求的Netty服务器也初始化完毕了，同时负责处理请求的线程池以及执行定时调度任务的线程池，也都初始化完毕了，可以说是，万事俱备只欠东风了

我们看下图



这个时候，就是最终要对BrokerContorller执行一下启动的逻辑，让他里面的一些功能组件完成启动时候需要执行的一些工作，同时最核心的，其实就是完成Netty服务器的启动，让他去监听一个端口号，可以接收别人的请求。

我们先回到BrokerStartup启动组件的main()方法中去，可以看看里面的内容：

```
public static void main(String[] args) {
    start(createBrokerController(args));
}
```

大家会发现上面的main()方法中，已经完成了BrokerContorller的初始化，接着就是执行了start()方法，于是我们进入start()方法可以去看看。

```
public static BrokerController start(BrokerController controller) {
    try {
        controller.start();
        String tip = "The broker[" + controller.getBrokerConfig().getBrokerName()
            + ", "
            + controller.getBrokerAddr() + "] boot success. serializeType="
            + RemotingCommand.getSerializeTypeConfigInThisServer();
        if (null != controller.getBrokerConfig().getNamesrvAddr()) {
            tip += " and name server is "
                + controller.getBrokerConfig().getNamesrvAddr();
        }
        log.info(tip);
        System.out.printf("%s\n", tip);
        return controller;
    } catch (Throwable e) {
        e.printStackTrace();
        System.exit(-1);
    }
    return null;
}
```

大家自己看看上面的start()方法，其实别的业务逻辑倒没什么，最主要就是执行了BrokerContorller的start()方法，也就是去自动了他

我们继续看，下面就是BrokerContorller.start()方法的源码了，大家仔细看里面我写的注释，都解释了每一个步骤是干什么的。

```

1 public void start() throws Exception {
2     // 这个还用解释么，一看就是在启动核心的消息存储组件
3     // 当然，这个时候消息存储组件在启动的时候，都干了一些什么，其实大家先不用管
4     if (this.messageStore != null) {
5         this.messageStore.start();
6     }
7
8     // 下面两个是非常核心的，他就是启动了Netty服务器，这样就可以接收请求了
9     // 至于说启动Netty服务器的源码，之前在NameServer那儿大致都看过了
10    // 如果大家对Netty的一些技术细节感兴趣，可以去网上查阅Netty的一些技术资料
11    if (this.remotingServer != null) {
12        this.remotingServer.start();
13    }
14    if (this.fastRemotingServer != null) {
15        this.fastRemotingServer.start();
16    }
17
18    // 下面是FileWatchService，是跟文件相关的一个服务组件的启动
19    // 暂时而言，我们先不用去关注他
20    if (this.fileWatchService != null) {
21        this.fileWatchService.start();
22    }
23
24    // 这个其实是比较关键的一个东西，这个BrokerOuterAPI是核心组件
25    // 这个组件实际上是让Broker通过Netty客户端去发送请求出去给别人的
26    // 比如说Broker发送请求到NameServer去注册以及心跳，其实都是通过这个组件
27    if (this.brokerOuterAPI != null) {
28        this.brokerOuterAPI.start();
29    }

```

```

31    // 下面几个都是实现功能的核心组件，暂时你不用过多的关注他
32    if (this.pullRequestHoldService != null) {
33        this.pullRequestHoldService.start();
34    }
35    if (this.clientHousekeepingService != null) {
36        this.clientHousekeepingService.start();
37    }
38    if (this.filterServerManager != null) {
39        this.filterServerManager.start();
40    }
41    if (!messageStoreConfig.isEnableDLegerCommitLog()) {
42        startProcessorByHa(messageStoreConfig.getBrokerRole());
43        handleSlaveSynchronize(messageStoreConfig.getBrokerRole());
44        this.registerBrokerAll(true, false, true);
45    }
46
47    // 下面是很关键的一个代码逻辑
48    // 这里其实是往线程池里提交了一个任务，让他去给NameServer进行注册
49    // 这个其实是下一次我们要讲的核心源码，我们去看看Broker启动后怎么去注册
50    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
51
52        @Override
53        public void run() {
54            try {
55                BrokerController.this.registerBrokerAll(true, false, brokerConfig.isForceRegister());
56            } catch (Throwable e) {
57                log.error("registerBrokerAll Exception", e);
58            }
59        }
60    }, 1000 * 10, Math.max(10000, Math.min(brokerConfig.getRegisterNameServerPeriod(), 60000)), TimeUnit.MILLISECONDS);

```

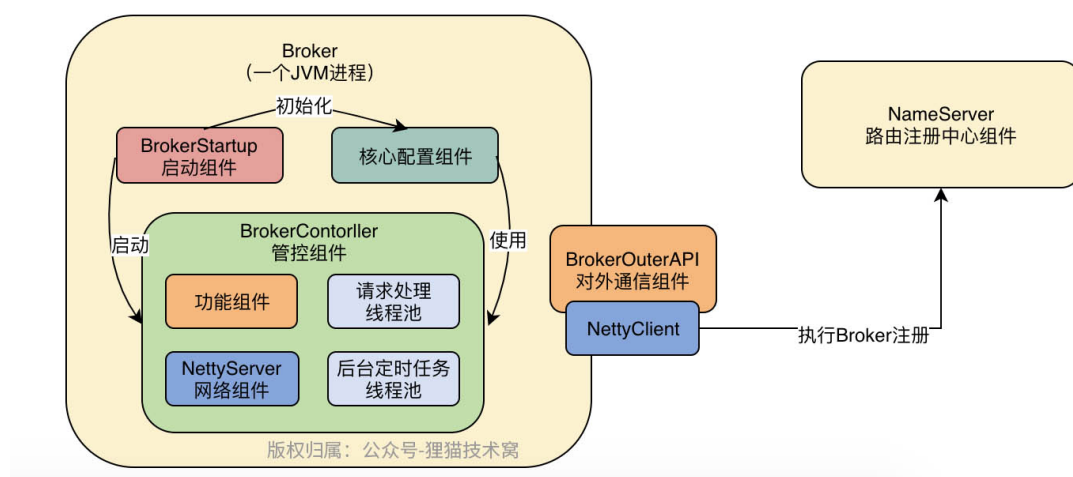
```

62 // 下面两个其实也是一些功能组件的启动
63 if (this.brokerStatsManager != null) {
64     this.brokerStatsManager.start();
65 }
66 if (this.brokerFastFailure != null) {
67     this.brokerFastFailure.start();
68 }
69
70
71 }

```

看完上述源码，大家其实从中只要提取一些核心的东西，知道说Netty服务器启动了，可以接收网络请求了，然后还有一个BrokerOuterAPI组件是基于Netty客户端发送请求给别人的，同时还启动一个线程去向NameServer注册，知道这几点就可以了。

在这里，我在下面的图里，就给大家展示出来了，BrokerOuterAPI和向NameServer注册这两个东西。



大家其实只要看完本篇文章，能理解到上图中的架构，就足够了，因为其实阅读和理解其他人写出来的复杂的源码，是一件很困难的事情，很多时候自己写的代码过两年都看不懂了，何况是看别人写了几年的代码呢！

所以其实看源码的时候，很重要的一个技巧，就是一定要有耐心，而且要抓住这个开源系统运行的主要流程和逻辑，从源码里重点抓住主要的一些组件和主要的流程，而不是在看源码的时候陷入各种组件的细节里去。

源码又不是你写的，你说你假设这个时候去看BrokerOuterAPI、RemotingServer、FileWatchService、MessageStore这些核心组件的源码细节，你觉得你这个时候看得懂吗？

只能说在看到现在这个程度的时候，你大致脑子里有个印象，你知道Broker里有这么一些核心组件，都进行了初始化以及完成了启动，但是你应该最主要关注的事情是这么几个：

- (1) Broker启动了，必然要去注册自己到NameServer去，所以BrokerOuterAPI这个组件必须要画到自己的图里去，这是一个核心组件
- (2) Broker启动之后，必然要有一个网络服务器去接收别人的请求，此时NettyServer这个组件是必须要知道的
- (3) 当你的NettyServer接收到网络请求之后，需要有线程池来处理，你需要知道这里应该有一个处理各种请求的线程池
- (4) 你处理请求的线程池在处理每个请求的时候，是不是需要各种核心功能组件的协调？比如写入消息到commitlog，然后写入索引到indexfile和consumer queue文件里去，此时你是不是需要对应的一些MessageStore之类的组件来配合你？
- (5) 除此之外，你是不是需要一些后台定时调度运行的线程来工作？比如定时发送心跳到NameServer去，类似这种事情。

所以当你从一个很高的角度去思考了Broker的运行之后，再切入到他的源码里，你会发现，其实你可以很轻松的从源码运行流程里提取出来一些核心组件，画到你的图里去，然后你的脑子里会轻易记住一个图。

迄今为止，上面那幅图，就是你对Broker源码的一个了解。

接着再往后走，一定要从各种场景驱动，去理解RocketMQ的源码，包括Broker的注册和心跳，客户端Producer的启动和初始化，Producer从NameServer拉取路由信息，Producer根据负载均衡算法选择一个Broker机器，Producer跟Broker建立网络连接，Producer发送消息到Broker，Broker把消息存储到磁盘。

上面我说的那些东西，每一个都是RocketMQ这个中间件运行的时候一个场景，一定要从这些场景出发，一点点去理解在每一个场景下，RocketMQ的各个源码中的组件是如何配合运行的。

千万不要在看源码的时候，就傻乎乎的一个类一个类的看，那样绝对是会放弃阅读一个源码的！

**End**

专栏版权归公众号**狸猫技术窝**所有

未经许可不得传播，如有侵权将追究法律责任

---

狸猫技术窝精品专栏及课程推荐：

[《从零开始带你成为JVM实战高手》](#)  
[《21天互联网Java进阶面试训练营》（分布式篇）](#)  
[《互联网Java工程师面试突击》（第1季）](#)  
[《互联网Java工程师面试突击》（第3季）](#)

---

**重要说明：**

如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑

如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群》（**购买后可见**）